#

Exception

Yulin XIE

####1.try/catch

In order to avoid predictable problems, we are forced to use try/catch for some commands like `FileInputStream`.

```
package exception;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;

public class TestException {

    public static void main(String[] args) {

        File f= new File("d:/LOL.exe");

        try{
            System.out.println("try to open d:/LOL.exe");
            new FileInputStream(f);
            System.out.println("succeed");
        }
        catch(FileNotFoundException e){
            System.out.println("d:/LOL.exe does not existe");
            e.printStackTrace();
        }

    }
}
```

We can surely use something like `catch(FileNotFoundException|ParseException e)` to include several excecptions at a time and in the block we then use something like `if(e instanceof FileNotFoundException){...}` to decouple the combined exception. Instead of using several `catch` after the `try`.

####2.Usage of `throws`

`throws` appears on the declation of a function to showcase a posibility of throwing an exception, which will not necessarily happen. On the contary, when we use `throw` we are indeed throwing an exception.

```
package exception;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;

public class TestException {

    public static void main(String[] args) {
        method1();

    }

    private static void method1() {
        try {
            method2();
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }
```

```
    private static void method2() throws FileNotFoundException {

        File f = new File("d:/LOL.exe");

        System.out.println("try to open d:/LOL.exe");
        new FileInputStream(f);
        System.out.println("succeed");

    }
}
```

In the above case, we throw an exception (if needed) in method2 and in we catch it in method2.

####3.try/catch/finally

It is NOT good to write codes like this because it will cause ambiguity to what will return in the end:

```
package exception;

public class TestException {

    public static int method() {
        try {
            return return1();
        } catch (Exception e) {
            return return2();
        } finally {
            return return3();
        }
    }

    private static int return1() {
        System.out.println("return 1");
        return 1;
    }
    private static int return2() {
        System.out.println("return 2");
        return 2;
    }
    private static int return3() {
        System.out.println("return 3");
        return 3;
    }

    public static void main(String[] args) {
        int result = method();
        System.out.println("result:" + result);

    }
}
```

The better way is as follows:

```
public static int method(){
  int result;
  try{
    result = return1();
  }catch(Exception e){
    result = return2();
  }finally{
    result = return3();
  }
  return result;
}
```

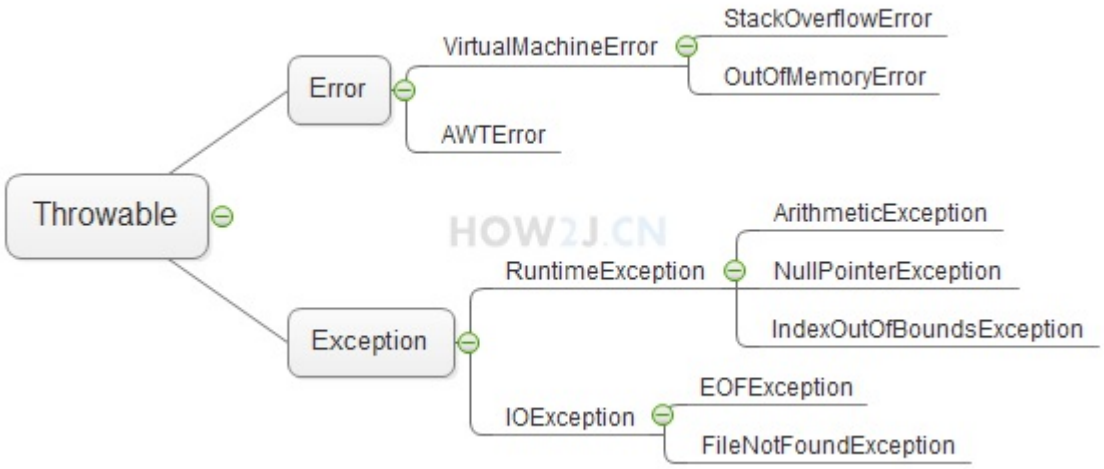
As for the two exemples aboved, we will go through `return1()` then `return3()`, but at the end it will return result of `return3()` as shown in the better way.

####4.Classification

1. CheckedException(e.g. FileNotFoundException,either *try/catch* either *throws exception* in declation)
2. RuntimeException
   (e.g. ArithmeticException / ArrayIndexOutOfBoundsException / NullPointerException, Not obligated to use try/catch)
3. Error(e.g. OutOfMemoryError)

> 运行时异常与非运行时异常的区别：
>
> - 运行时异常是不可查异常，不需要进行显式的捕捉
> - 非运行时异常是可查异常，必须进行显式的捕捉，或者抛出

####5.throwable
throwable is the father class of error/exception. It's possible to throw a `throwable` in a function but it will also bring ambiguity because we have no idea it will be which kind of problem, so try to be specific about the exception.



####6.Self-defined exception
We can define our personalized exception by inheriting class exception, which may look like this:

```
package com.company;
import org.w3c.dom.ls.LSOutput;
import javax.crypto.spec.PSource;
class hero{
    class heroDeadException extends Exception{ //self defined exception
        public heroDeadException(){}
        public heroDeadException(String msg){
            super(msg);
        }
    }
    String name;
    int hp;
    public hero(String name, int hp){
        this.hp = hp;
        this.name = name;
    }
    public void attack(hero another) throws heroDeadException{ //throws exception
        if(another.hp<=0){
            throw new heroDeadException(another.name+" is dead, no more attack");// here we throw out exception
        }
        another.hp-=1;
        System.out.println(another.name+" rested hp:"+another.hp);
    }
```

```
}
public class Main {
    public static void main(String[] args) {
        hero a = new hero("a",99);
        hero b = new hero("b",2);
        try{
            while(true) {
                b.attack(a);
                a.attack(b);
            }
        }catch (Exception e){ // here we need to catch exception
            e.printStackTrace();
        }
    }
}
```